A GENERAL SOFTWARE MODULE FOR CAMAC, EQUIPMENT AND COMPOSITE VARIABLE CONTROL

A. DANEELS and P. SKAREK

PS Division, CERN, 1211 Geneva 23, Switzerland

Current PS controls application software has a strong hierarchical structure of software modules which translate user-friendly commands into the intricacies of hardware devices. Starting from the most elementary hardware level, these are: (i) the “interface module” (IM), which hides the various CAMAC commands so as to provide a standard access to each type of CAMAC module; (ii) the “equipment module” (EM) presents a simple standard software interface of each process equipment. There is one EM for each type of equipment and all process equipment is accessed through EMs via IMs; (iii) the “composite variable module” (CVM) provides control of abstract beam variables. It involves control of several and possibly different kinds of equipment. Setting a CVM results in appropriate setting of all relevant equipment through calls to their EMs.

For the LEP Preinjector (LPI) a new generation of application software is being implemented based on the experiments with the current system and the technological evolution since its conception. The logical levels of IM, EM and CVM are kept, but their managerial and housekeeping functions are merged into a single module: the “General Module” (GM).

This paper represents the characteristics of the GM, its decomposition into housekeeping and management activities ones to supervise device specific controls, and its structuring into logic and data modules. This results in a unique frame for all modules in the application hierarchy. It is transparent to the programmes so that the development of specific IMs, EMs and CVMs reduces to editing appropriate data tables, developing specific codes or re-using existing ones. This new generation of application software follows modern ideas on conceptual modeling by data abstraction and object-oriented programming.

1. Introduction

The CERN PS accelerator complex comprises several proton accelerators, an antiproton accumulator and various beam transfer lines which are controlled from a central control room through a network of about 20 minicomputers and 100 microcomputers interfaced to the process hardware through CAMAC [1]. This network comprises a message transfer computer, dedicated process computers (FECs, front-end computers) controlling subsystems of the accelerator, and console computers driving the main consoles. The microcomputers are located in the CAMAC crate as auxiliary crate controllers (ACCs).

The accelerators provide beam pulses with characteristics changing from cycle to cycle in a recurring sequence called a supercycle. This modification of beam characteristics from one cycle to another is called pulse-to-pulse modulation. The sequence of these pulses is defined at any of the main consoles. Operators in the main control room may tune these beams concurrently as if they were provided by independent accelerators.

In view of the LEP preinjector controls, the present system is being extended and upgraded taking into account the experience and technological evolution since its early conception in 1974. The major objectives of this new generation application software are improved reliability, maintainability and application program development productivity, while staying compatible with the existing facilities [2].

The software modules which are discussed, i.e. IM, EM and CVM, are all located in the FECs. The control originates either from application programs in console computers, transmitted by remote procedure calls as shown in sect. 3.1 or from programs running directly in the FECs. For the LEP preinjector software these modules will even be running in the microprocessors in ACCs [3]. The application program addresses a certain FEC, since a FEC is specific to a certain subsystem of the accelerator; but it need not know in which microprocessor the GM is located.

2. The hierarchy of control functions

2.1. Different levels and different modules

Whether the control software is located in the front-end computer (FECs) or in the microprocessors, the lowest level it sees is CAMAC, a collection of hardware modules. Interface Modules (IM) are the corresponding software modules which control these CAMAC modules.

The next software level is Equipment Modules (EM): i.e. subroutines which give the operator in the main control room the same facilities for executing remotely
those functions he could perform by local, manual control, e.g. turning knobs, switching on and off, reading meters, etc. Through the EM he controls the equipment by software calls using parameters relevant for operation: hardware intricacies are masked. This is based on the old idea of an equipment handler. There is one EM for each type of equipment, e.g. one for stepping-motors, rf-cavities, power supplies, etc. The EM smooths out and hides small differences in hardware from the application programs. Any change of hardware does not effect the application programs, but only some data in the EMs.

An EM calls CAMAC via the standard IM sub-routines. There is one IM for each CAMAC module: input/output registers, analog-to-digital converters, etc. The IMs promote a better integration of hardware and software design. For severe time-critical applications CAMAC is accessed directly via an appropriate RT-program in the micros. In the new LPI software structure these RT-programs also access CAMAC via IMs.

The next level up again are Composite Variable Modules (CVM) that model of control of beam variables. A composite variable \( V_c \) is defined as a combination of \( n \) individual process variables \( V_1, V_2, \ldots, V_n \). The control value to which such a variable \( V_c \) is set, results from a mathematical relation between the control values of each individual component. Similarly, the acquisition of this variable results from the acquisition of a number of individual components which need not be identical to the individual components for control.

The application software sees CVMs as an EM which calls other EMs. They have the same calling sequence as EMs, but setting the CVM results in appropriate settings of relevant equipment through calls to their EMs. The CVM provides an additional level of abstraction: it gives the operator a clearer view of what he is controlling, and it allows one – on the software side – to keep the EM-concept clean and simple.

2.2. An example

As an example we discuss briefly the Composite Variable Module named the CVM-SBP which controls closed orbit deformations of the beam through a combination of bumper magnets. At the injection into the PS, the closed orbit is deformed so that the incoming beam may join it at a specific angle and position. This deformation is produced by setting proper currents in four magnets via the Equipment Module EM-POW. These power supplies have to be set and their current measured at very specific instants in the accelerator cycle. Therefore timing counters have also to be controlled through another EM, EM-PTIM, CVM-SBP sets and acquires angle and position values and checks whether these values are realistic, i.e. will the beam stay within the vacuum chamber, can it be achieved within the range of currents obtainable by the power supplies, etc.

After having calculated control values for the power supplies it transfers them to EM-POW, and necessary timing values are sent to EM-PTIM. Now the EMs take over and finally delegate their last job – sending the corresponding CAMAC bit-pattern to the equipment – to an RT-task or call directly the IMs (IM-QUADTRANSCIEVER for EM-POW and IM-PRESETCOUNTER for EM-PTIM). For our new LEP pre-injector layout, all these modules reduce to just one, the GM, and all the peculiarities of CVM-SBP, EM-PTIM, EM-POW, IM-QUADTRANSCIEVER and IM-PRESETCOUNTER reside in data tables.

3. Layout and internal structure of the GM

3.1. The interface to application programs

The GM presents a standard and unique interface to the user (i.e. application programs or the NODAL interpreter) for all modules (IM, EM, CVM and any other that can be invented), whatever their type or their position in the logical hierarchy. Two types of access are provided:

**Single entry:** to execute a single function on a single equipment:

\[
\text{GM (Module-type, Value, Flag, Elno, Function, Beam-Id, Compl)},
\]

**Array entry:** to execute a single function on a large set of equipment of the same type:

\[
\text{AGM (Module-type, Value-Array, Flag, Elno-Array, Function, Beam-Id, Compl-Array)}.
\]

Short explanation of the parameters:

- **Module-type:** Serial number of the general module or just its name (e.g. POW, PTIM).
- **Value:** Controlled or acquired value, or any value from tables.
- **Flag:** Indicates read or write.
- **Elno:** Element number: sequential number of the module to be controlled.
- **Function:** (also called Property for historical reasons): a shorthand notation for the control function to be executed (what to do with the equipment addressed), e.g. CCV for current control value, AQN for acquisition, POWER for switching on/off, etc.
- **Beam-Id:** identifies the type of beam within the supercycle sequence for which the requested function is performed. (This parameters takes the possibility for pulse-to-pulse modulation into account.)
- **Compl:** Completion code and error indicator.
3.2. Internal structure of the GM

The general code in a GM is unique and consists of the GM-kernel (see fig. 1) and the datatable access routines (DTARS). All equipment specific individual information is contained in the tables: the datatables (DTs), the array builder tables (ABTABs) and the property branch tables (PBTs). If the control cannot be completely table-driven, then the module's specific logic is contained in property codes (PROCOS). Modules can share PROCOS.

3.3. Functional description of the GM

The GM-kernel checks the validity of the input parameters (e.g. if the element number is within the allowed range) and calls the general array builder (GAB), which is the heart of the GM [4].

The GAB has two main functions:

1) It finds out via the property branch routine (PBR) by using the data in the Property Branch Table (PBT), which property code (PROCO) to call, depending on the module type and the requested property in the input parameter list.

2) It builds up an array call to the lower level module: e.g. on the CVM-level, a call to all related EMs is constructed and executed. The call to the lower level module is done via a recursive call to the GM. All necessary information, as to what lower level module and its related element numbers is kept in the array builder tables (ABTABs). In the previous example, section 2, the ABTAB for CVM-SBP tells that the EM-POW and EM-PTIME are concerned, with four POW element numbers, etc.

The Datatable (DT) contains equipment specific data and process control/acquisition values, for example, the current control values, and also less frequently changing data as limit values, scaling factors etc. Access to the DTs is via standard routines DTARS. These access routines have two additional functions. Since the DTs are global structures (non-reentrant), write-access is subject to a reservation scheme. The DTARS limit write-access to users which have reserved first. Reservation is also used to coordinate access to devices which share common hardware by defining the element numbers just as different access channels to the hardware.

To summarize: the GM is designed to ensure a clean separation between general program logic and data. The control logic for a given module is fully table-driven by the three tables DT, ABTAB and PBT. If an additional control logic is needed, it is localized pieces of code, the PROCOS, which are executed selectively as indicated in the PBT. Technically speaking, the GM is a subroutine and has no influence on external conditions or timing requirements when to execute. It is the job of the calling main program, the RT-task, to call the GM when needed.

3.4. Framework programming

In the current control system where CVMs, EMs and IMs are still individual entities, there is a clear separation between housekeeping functions, general control functions and data. The general code is contained in a “Frame”, which is a file of standard software which can be edited. It is seen as a form the programmer has to fill in. He modifies this file by the inserting code and data at well defined locations, and by deleting parts that are not relevant.

This framework has proved very valuable not only for writing EMs and CVMs, but also for maintaining and understanding such modules written by other programmers: It speeds up production, produces more reliable and standard software, and allows specific aspects of the CPS control system (e.g. pulse-to-pulse modulation, reservation, etc.) to be fully transparent to the application programmer.

The new layout based on the GM goes a step further in the direction of automating the software development process. Through special offline editors, the programmer fills the various data tables to define the equipment he needs to control, whereas real coding is restricted to those properties which are more specific and for which no code exists yet.

Online updates may be confusing and dangerous. Special editors are provided to update master files in the central data base, which are then downloaded into the central data base.
target microcomputers. These editors ensure all necessary consistency checks and safeguard against erroneous operations.

4. Conceptual aspects of the GM

4.1. Module strength and module coupling

Myers’ Composite/Structured Design [6] gives methods for modular software design which allows measurement of the quality of a modular decomposition. The degree of interdependence among the major pieces of a programming system, i.e. the general complexity, should be reduced as much as possible. The module strength, i.e. the internal relationships within a single module, should be as high as possible. The module coupling, the relationships among all modules in a system, should be as low as possible. Myers’ ideas and terminology have been applied to evaluate the decomposition of the CPS control application software system. The detailed investigation can be found in [5], here we give only the results.

Myers defines logical strength modules as modules performing a set of related functions, one of which is explicitly selected by the calling module. The interface to a logical-strength module includes some type of function.

Though IMs, EMs and CVMs seem to have logical strength, a closer look reveals that they rather have informational strength. According to Myers, the highest strength a module can have is functional: i.e. when the module performs a single function. This is typically the case for the control functions in property codes. However, according to Myers again, modules have a maximum of independence when they achieve informational strength: i.e. multiple related function strength modules (i.e. the PROCOs) packed together to hide information. The hidden information is kept in the data structures, and the resource hidden is the hardware with all its nonstandard behaviour and exceptions. This is typically the case for IMs, EMs and CVMs.

EMs have been defined as modules reflecting a category of equipment. Effectively the coupling between EMs is nil: they can be considered as the basic building blocks in the software structure. But also the coupling to the application program is very weak. It is data coupling with all interface data in the form of nonglobal homogeneous data items (formal parameters).

In addition to EMs, CVMs which reflect beam variables result from relating different equipment both functionally and informationally. In software this is achieved by coupling modules, i.e. by packing logically together into one module two or more modules now having informational and functional relations. These CVMs have therefore similar high strength (informational) and low coupling (data) as EMs. CVMs were chosen to have exactly the same interface (same calling sequence) and the same logic and structure as the EMs, except that they call one or more EMs. The EM-concept has thus been applied recursively as a natural consequence of our structure.

The same discussion can be repeated for the IM: therefore it also applies to the GM in general.

4.2. Object oriented programming

The GM can also be viewed differently. In the object oriented programming paradigm [7] a physical entity is modelled by an object; objects are instances of classes and can be defined in an inheritance hierarchy. Objects communicate by sending message to each other. Objects consist of information (private data belonging to the object) and procedures manipulating that data (called methods). A certain message name is called a selector. If one wants an object to react in a certain manner, one sends a generic message to the object and its method responds to the message in an object specific way.

The physical entities (CAMAC modules, equipment and beam variables) are modelled by objects belonging to classes like CVM, EM and IM. For example EM-POW is just an instance of the class EM, its instance variables are kept in the data table. The GM can be considered as a superclass from which the classes CVM, EM and IM inherit the general control mechanisms. The generic message sending is covered by the property codes. Property names are the selectors for the methods, the PROCOs. Each individual module can respond differently to the generic message, since it can implement different methods (different PROCOs for the same property). From outside, there is just one generic method: “send the CCV”, but hidden in the objects, there are different methods.

In the jargon of the object oriented approach we can say: The GM is a superclass, covering the CMV-, EM- and IM-classes. Instance variables of the objects are the datatables and the protocol defining the response to generic messages (our properties) are defined by methods, implemented as the property branch table and the PROCOs.

4.3. Language issues

Usually equipment drivers are programmed in assembly code. However, without a high level language supporting some sort of security for abstract data types [8], the advantages of the module design: easy understanding, maintenance and re-usability would be illusory. Therefore the GM – whether located in the FECs or in the micros – and with all levels down to CAMAC access, is implemented in a PASCAL-like language [9]. This language supports abstract data types by
providing information hiding and locality of data access, as well as separating abstract properties from implementation.

The whole GM can be considered as an abstract data type in itself: the GM is merely a datatable (a data structure) with an access mechanism which is transparent and hidden in the GM and the datatable access routines.

5. Conclusion and outlook

The software of the CPS control system is built of layers which represent different levels of abstraction. The GM which is being implemented in view of the LPI controls follows the natural evolution in modern process control: from the software’s point of view there is basically no difference between controlling a CAMAC module, a process device or a beam physics parameter. They all need the same basic functions: housekeeping (like validity check against limits, reservation check for conflicting demands, etc.) and access to data tables.

In the new structure the property code is considered as the elementary building block. They represent the atomic functions for control as they cannot be broken down further without losing their physical significance.

All control can be described in terms of data and logic of some standard format. It is shown that the way the boundaries between data and logic are defined results in a very general and powerful software module which can be applied to any level in the controls software structure.

Acknowledgments

We gratefully acknowledge the support of our group leader, Berend Kuiper, and valuable contributions from our colleagues L. Casalegno and C.H. Sicard.

References